TRW Systems Engineering &
Development Division

TRW-TS-89-01

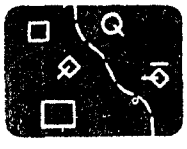
TRW

Software Project Management Using Effective Process Metrics: The CCPDS-R Experience

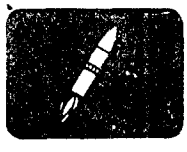
Don Andres

November 1989

TRW Technology Series



TRW Technology Series

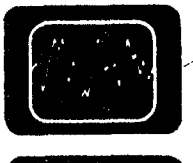


TRW Technology Series

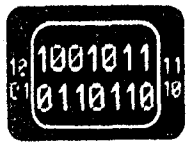
TRW Technology Series

TRW Technology Series

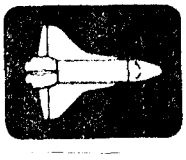
TRW Technology Series



TRW Technology Series



TRW Technology Series



hnology Series

SOFTWARE PROJECT MANAGEMENT USING EFFECTIVE PROCESS METRICS: THE CCPDS-R EXPERIENCE

Donald Andres
TRW Defense System Group
Redondo Beach, CA

ABSTRACT

The paper captures the project management experience of using process metrics to measure the development of a large software system for the U.S. government. The Command Center Processing and Display System-Replacement (CCPDS-R) program is a development effort consisting of three subsystems and over 600,000 lines of Ada source code executing in a distributed Digital Equipment Corporation VAX VMS environment. The initial subsystem called the Common Subsystem, consists of over 300,000 lines of Ada source code and is 2 years into development with a successful Critical Design Review (CDR) held at month 25 of the contract. Utilizing the new Ada process model, over 280,000 lines of Ada source code have been developed and integrated into a working demonstration on the operational hardware. This approach has permitted the validation of the software design, allowed early analysis and resolution of the software performance issues, and provided extensive insight into the usability and flexibility of the system for the user community.

This first program use of the incremental, demonstration based process model with related software metrics has been totally successful. The technology now is being transferred to other projects within the company. The U.S. Government acquisition agency has designated CCPDS-R as its "flagship program" and is basing new acquisitions on the approach utilized by the CCPDS-R program.

PROJECT BACKGROUND

The Command Center Processing and Display System Replacement (CCPDS-R) program will provide display information used during emergency conferences by the National Command Authorities; Chairman, Joint Chiefs of Staff; Commander in Chief North American Aerospace Command; Commander in Chief United States Space Command; Commander in Chief Strategic Air Command; and other nuclear capable Commanders in Chief. It is the missile warning element of the new Integrated Tactical Warning and Assessment system architecture developed by North American Aerospace Defense Command/Air Force Space Command.

The CCPDS-R program is being procured by Air Force System Command Headquarters Electronic Systems Division (ESD) at Hanscom AFB and was awarded to TRW Defense Systems Group in June 1987. TRW will build three subsystems. The first, identified as the Common Subsystem, is 25 months into development. The Common Subsystem consists of over 300,000 source lines of Ada with a softare development schedule of 38 months. It will be a highly reliable, near real-time distributed system with a sophisticated User Interface and stringent performance requirements. It will be implemented entirely in Ada.

The approach to software development on CCPDS-R is unique to the industry. Extensive tailoring of DoD-STD-2167, by TRW and the government as a team, allowed many innovative techniques and the orchestration of a successful Ada software development program. These innovations already have been proposed on other TRW developments and could well become a DoD and industry standard for Ada software development.

The specific features of the new Process Model are described in another paper, "TRW's Ada Process Model for Incremental Development of Large Software Systems" [Royce 1990]. In summary, the Ada process Model is a uniform application of incremental development coupled with a demonstration-based approach to design review for continuous and insightful thread testing and risk management. The use of Ada as the life cycle language for design evolution provides a vehicle for uniformity and a basis for consistent software progress metrics. Ada has proven also to be a significant benefit in the incremental development/integration/test approach, particularly in enabling rapid integration of software from multiple CSCIs.

The CCPDS-R Software test approach is described in a paper, "Incremental Software Test Approach for DoD-STD-2167A Ada Projects" [Springman 1989]. The test approach has been modified and enhanced significantly as both TRW and the customer better understand the test requirements and implications of specific test techniques. Tailoring of the approach is necessary as experience is gained from earlier test phases.

This paper describes the management implications of operating under the new software development process. The primary emphasis and features provide early visibility into the process with extensive prototyping, incremental builds and early demonstrations. The process is then monitored and managed by use of software process metrics tailored to the CCPDS-R program.

o Management Characteristics

CCPDS-R employ a number of innovative management and technical methods that are intended to maximize software development productivity while minimizing development and test risks. Many of these methods take advantage of Ada software engineering techniques, and depart from the traditional software development "waterfall" process in favor of an approach closer to the "spiral" model [Boehm 85]. The CCPDS-R approach emphasizes continuous early design integration and demonstration (Figure 1). TRW has incorporated the Ada Process Model in the CCPDS-R software planning documents, including the Software Development Plan (SDP) and the Software Standards and Procedures Manual (SSPM). The model is based on incremental development and extensive prototyping, and is being refined as experience is gained during the development of the CCPDS-R software. The particular innovations of the CCPDS-R approach are discussed briefly herein. These innovations have helped formulate a good set of process metrics to measure software status.

o Two Pass Design Approach

The CCPDS-R approach is a two pass design philosophy (Figure 2); the first pass is prototyping to demonstrate feasibility and the second pass formally captures the design recommended by the prototyping results. To employ this approach most productively, easily reconfigurable software architecture components are needed to rapidly integrate applications components to determine software functionality and projected performance. On CCPDS-R, these components

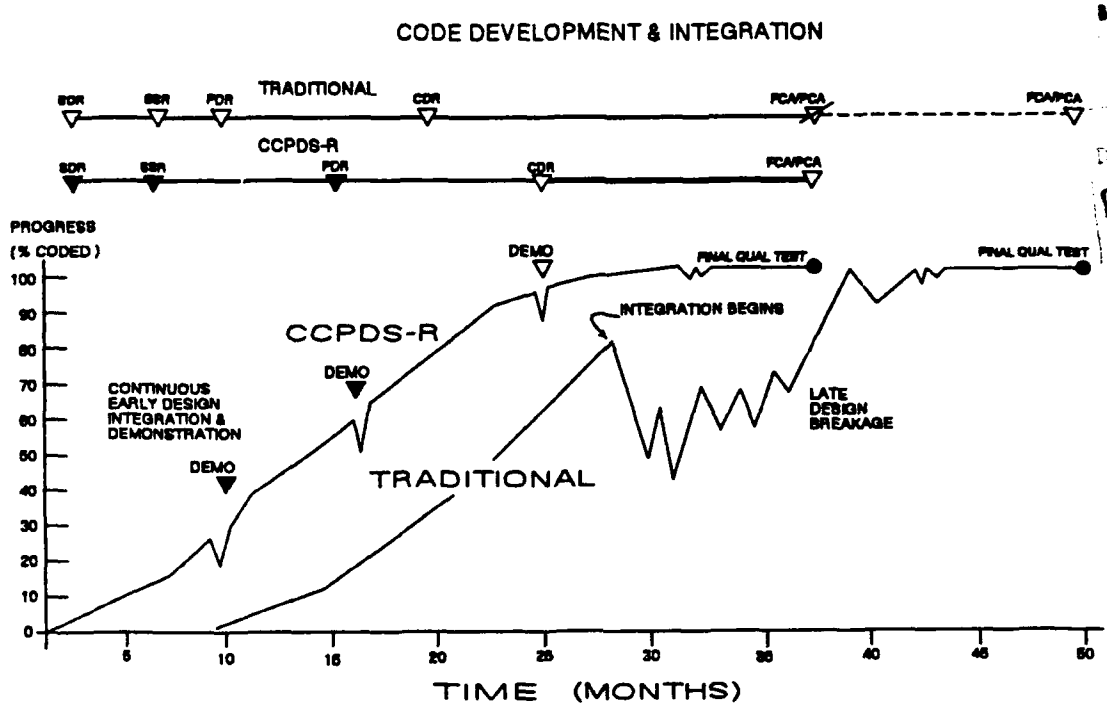


Figure 1: Common Subsystem Software Development and Integration

are provided by the Network Architecture Services (NAS) CSCI and the Software Architecture Skeleton (SAS) which encapsulates the top level executable components and their interfaces [Royce 89].

The management ramifications of this approach are associated with time and technical risk. It generally takes longer to complete the design and code phase, generate the related documentation, and prepare the software for "turnover" to the I&T group. This time is offset, however, by lowering the technical risk of undetected design problems and performance issues. The sooner problems are discovered, the cheaper they are to fix. A design flaw uncovered as a result of prototyping should not be regarded as a "failure", but rather as a positive result that will steer the design in the proper direction much earlier. The management focus of this new process is to identify and resolve all major software risk issues by PDR.

The incremental development approach has numerous management benefits; it breaks up a large software development job into manageable entities. This minimizes the communication complexity between team members since fewer individuals are working in the development of any one buld. Management must focus on contents of the builds initially to ensure cohesion in contents and then that functional capability (and correspondingly the development effort) does

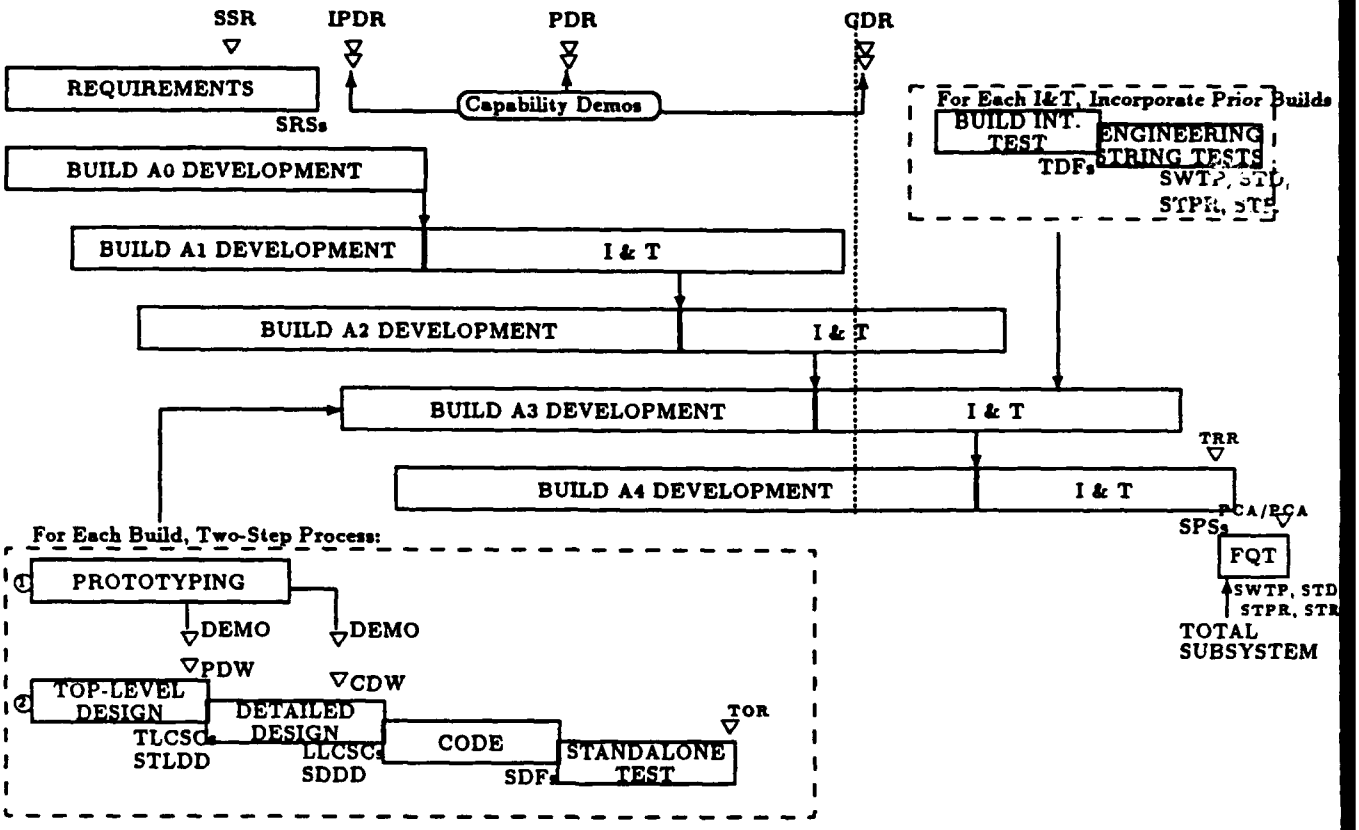


Figure 2: Software Development Approach-Common Subsystem

not slip to later builds. An indirect effect of this process is a highly motivated software engineering and development organization. They strive to produce an excellent product because of the near term visibility of their product - software code.

o Early Demonstrations

In concert with the early prototyping philosophy, tangible design progress is shown to CCPDS-R program management and the government early and often during the design process (Figure 2). On past programs, where design progress was measured by how much documentation and paper analysis was produced in support of PDR and CDR, the paper reviews were usually judged successful, with little visibility into design feasibility or potential problems. To correct this, formal demonstrations are scheduled for the Customer and Users at key points in the design process. Items to be demonstrated include user/system interfaces, functional software capabilities, and prototyped solutions for high risk items. The demonstrations are natural extensions of design activities, minimizing the generation of formal documentation and demo-specific software. These demonstrations provide early visibility in selected design areas so as to obtain early feedback, not to verify requirements or performance. Management attention is required to ensure that the government and the contractor stick to the purpose of the demonstrations. This approach has resulted in 125,000 source lines of Ada code being developed, integrated, and demonstrated at PDR (month 16 of current contract) and 280,000 Ada source lines at CDR (month 25). These demonstrations included critical components executing under peak load to provide tangible evidence that TRW's design would meet the stringent CCPDS-R

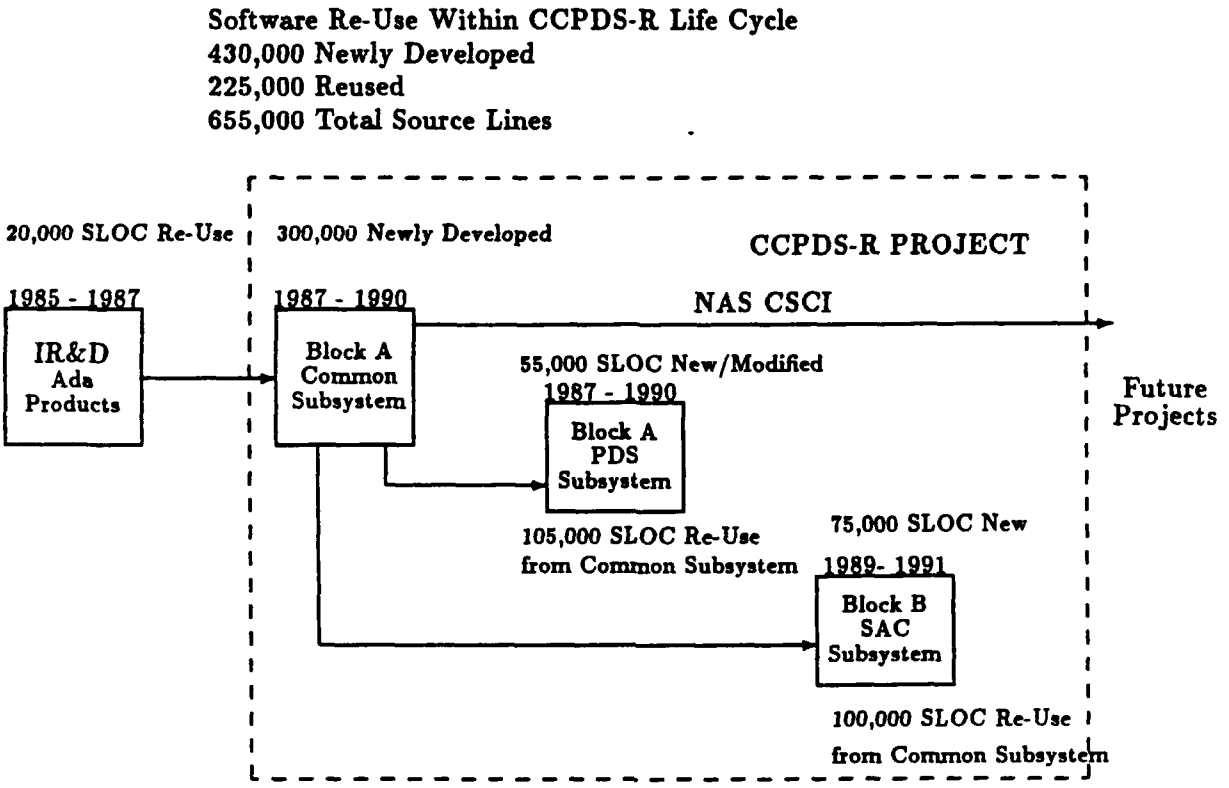


Figure 3: Reusable Software

performance requirements. Informal demonstrations are given as part of design walkthroughs or status reviews. The demonstration audiences include senior engineering personnel from other programs in TRW to promote technical feedback and reuse of by other programs.

o Reusable Software Components

Ada encourages modularity in the software, which promotes reuse of components. For example, on CCPDS-R, the entire NAS CSCI is essentially an application-independent set of components reusable by any DEC VAX/VMS application [Royce 1989]. On CCPDS-R, the software developed under the basic contract for the Common Subsystem will be heavily reused for subsequent Processing Display Subsystem (PDS) and SAC Subsystem options, because the basic software is being designed for reuse. (Figure 3)

Particular management attention is necessary to ensure the maximum reusability of the software. While this is aided by standards and features of Ada, the concept of reusability must continually be a topic for discussion at technical interchanges and design reviews. The government and development team must be kept aware of the reusability benefits so as to tailor the various subsystems requirements to enhance the potential for reusable software.

o Use of Schedule Tools

CCPDS-R has numerous milestones and hierarchies of schedules, which are used by management and developers at all levels of the program. It is essential to employ a scheduling methodology that is tool based in order to keep the process of regularly statusing and updating the schedules manageable. CCPDS-R is using the VISION scheduling system with a companion graphics generator. The scheduling organization has integrated all of the C/SCSC data, WBS elements, and detailed activity networks and milestones into a tiered set of schedules that satisfies Government and TRW needs. All schedules are statused monthly with the earned value cost data driving the schedule update process relatively automatically. This provides an easy mechanism to compare cost/schedule data with other process metrics. In addition, detailed weekly schedules are generated showing activities, milestones and products for each performer. These are statused at least monthly, and serve to create schedule awareness throughout the program to the worker level.

o Risk Management

Software risk management is a continuous activity on any large program. CCPDS-R's incremental development approach combined with extensive early prototyping is a continuous risk management strategy. In general, every program should generate a Risk Management Plan that identifies risks and concentrates on the top problem areas and mitigation procedures to be followed. CCPDS-R has a Risk Review Board, comprised of representatives from all program areas, that meets at least monthly to identify, assess and resolve risk items. All risk items are identified as early as possible, documented, and assigned to a specific individual to work, along with a specific plan for mitigating the risk. We regularly review the risk items to assure that all possible avenues for solutions are explored.

SOFTWARE DEVELOPMENT MANAGEMENT INDICATORS (METRICS)

A set of management metrics has been developed by TRW and the government to facilitate greater insight into the software development process. This is a direct fallout of a government directive by General Skantz requiring use of software process metrics. The CCPDS-R metrics were planned and implemented to minimize productivity impacts. Metrics collection standards were incorporated into the Software Standards and Procedures Manual early in the program (Month 3) to impose metrics collection through software standards. Automated tools were developed for collecting many of the detailed metrics directly from the evolving Ada design files.

One of the key features of the homogeneous Ada life cycle representation (i.e., Ada as a design language) [Royce 1990] is the consistency of metrics. Throughout the life cycle, the same metrics can be evaluated to ascertain progress, product change and product quality. Since Ada program units are being developed in the design phase, their metrics can be compared against planning estimates to evaluate progress and trends. This gives management objective insight early in the life cycle and allows reaction to any changes which are considered off-nominal before they result in undesirable side effects.

By providing continuous Software Problem Report (SPR) metrics against all configuration controlled components, component reliability and quality can be assessed. This provides early indicators of which components require further work. These metrics are tracked during the management of the project, with planned value profiles measured versus thresholds which

alert management that redirection/further evaluation is necessary. Under configuration management, design problems and fixes are formally tracked. The record of problems and solutions provides substantial data for predicting future performance and product reliability.

The following is a list of metrics which are gathered and reported monthly on the CCPDS-R project.

- CSCI and Subsystem Characteristics
    1. Number of Ada source lines (Figure 4) by CSCI and total subsystem
    2. Number of Software Development Files (SDFs) (Figure 6)
    3. Complexity
- Development Progress by Build, by CSCI and for the whole subsystem (Figure 6 and Figure 7)
    1. Design Progress: $\frac{Ada}{Ada+ADL}$ ratio
    2. KSLOC Standalone tested
    3. KSLOC Integrated
- Effort Expenditure
    1. Cost variances, Schedule variances (By CSCI and Build)
    2. Effort expended by Development activity (Preliminary Design, Testing, planning, etc.)
    3. Staffing (Actual vs Planned) (Figure 8)
- Software Architecture Stability
    1. Number of Nodes, Ada Main Programs, Ada Tasks, Sockets (Top Level Message Interfaces)
- Program Volatility (ECP Impacts)
- Action Item Closure History
- SPR History: Number of SPRs by Build, by CSCI and for the whole system (Figure 11)
- Software Testing
    1. Subsystem test progress (Figure 9)
    2. Number of test procedures (estimated and completed)
    3. Number of SRS requirements (existing and verified) (Figure 10)
- Reliability (Software failures following a baseline turnover) (Figure 12)
- Resource Utilization
    1. Host development environment
    2. Target hardware utilization

o Management Use of Metrics

Our experience with metrics collection and reporting has been very positive. Trends and issues have been explicitly observable, stimulating detailed questions and early resolution of potential problems. The metrics approach has also promoted more uniform communications, checks and balances and quantitative explanations for:

1. Customer/Contractor interaction
2. Customer/Higher Government Authority Discussions
3. Project/Corporate Management Review
4. Internal Software reviews
5. Software/Project Manager Review

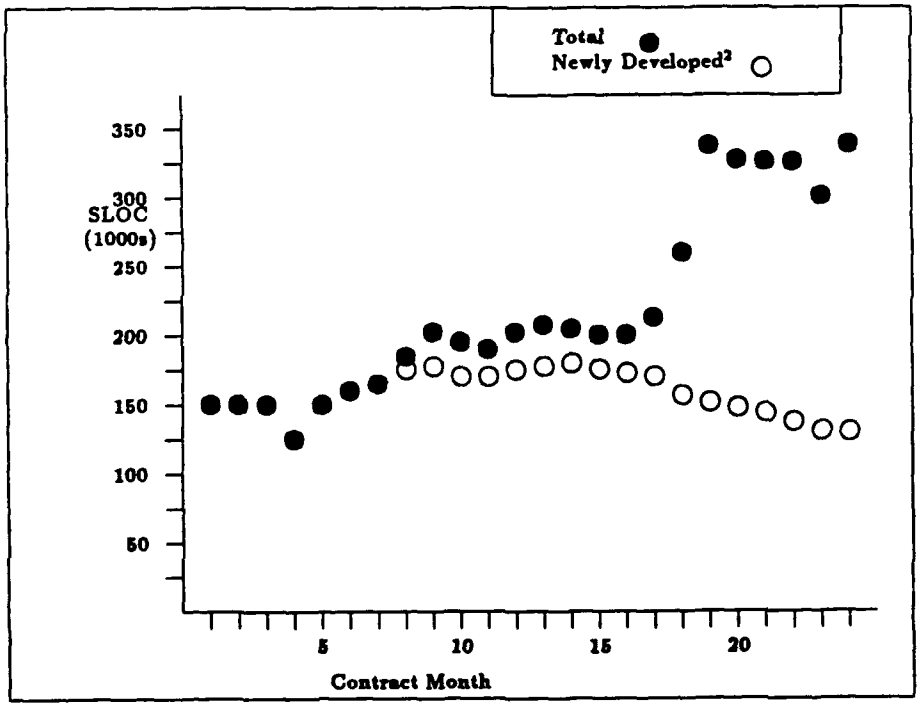
Highlights of these metrics will be discussed herein concentrating on the current progress (Month 24) of the program to illustrate the effective use of the metrics as management measurement tools.

The software size history (Figure 4) depicts the software size in terms of Ada source lines of code at contract award versus the current size estimate. The graphic depicts the estimate of the software size as it has changed based on monthly estimates. Normally, a graph as illustrated in Figure 4 would indicate disaster since the size has doubled in two years. However, on CCPDS-R, we adopted the Ada COCOMO method of counting source lines of code midway through the process [Royce 1990] and [Boehm/Royce 1988]. This resulted in the largest increase in code size - primarily in SSV. Secondly, we opted to produce some of the mundane, clerical software with tools as opposed to the brute-force coding technique. As a result, a software tool set sized at about 15,000 Ada source lines of code produces approximately 200,000 source lines of operational code (Table 1).

Tool Characteristics			Operational SW Produced		
Name	SLOC	Dev. MM	CSCI	Function	SLOC
SAS Builder Tool	6500	10	SSV	SW Architecture Skeleton	20,000
Format Build Tool	2500	6	DCO	Display Format Tables	17,000
CCPDS-R Message Tool	3500	12	CCO	External I/F Validation	23,000
				Format Outbound Messages	17,000
			TAS	Message Database Definitions	5,000
			SSV	SGI Message Type Declarations	
				SSMS	2,000
				Common & SAC	49,000
				CCOMS	3,000
				Forward Users & Sensors	6,000
Message IO Tool	2100	2	SSV	Data Reduction/Msg Print	60,000
Total	14,600	30		"Cookbook" Software	202,000

Table 1: Tool Produced Software

CSCI	Source lines (1000s)		
	Contract Award	Current Size	Auto[1]
NAS	20	18.6	0
SSV	18	182	140
DCO	47.5	40.4	17
TAS	17	9.6	5
CMP	23	10.2	0
CCO	24	75.5	40
Total	149.5	336.3	202



[1] Source Code Generated Automatically by Tool

[2] Newly Developed = Total Developed - Tool Generated

Figure 4: Development Progress-Common Summary

The overall progress for each subsystem is described at a high level via the Development Progress Summary (Figure 5). This single metric shows the development schedule, software size and progress to date by software build. The progress is depicted as percent complete for design/code and standalone test activities. The shading indicating overall build progress represents a combination of the financial cost/schedule data and the software detailed development metrics data. This metric gives a good overall picture of status and potential problem areas requiring management attention, and has been an effective means of conveying status to higher levels of management. The next level of detail for software development progress indicates

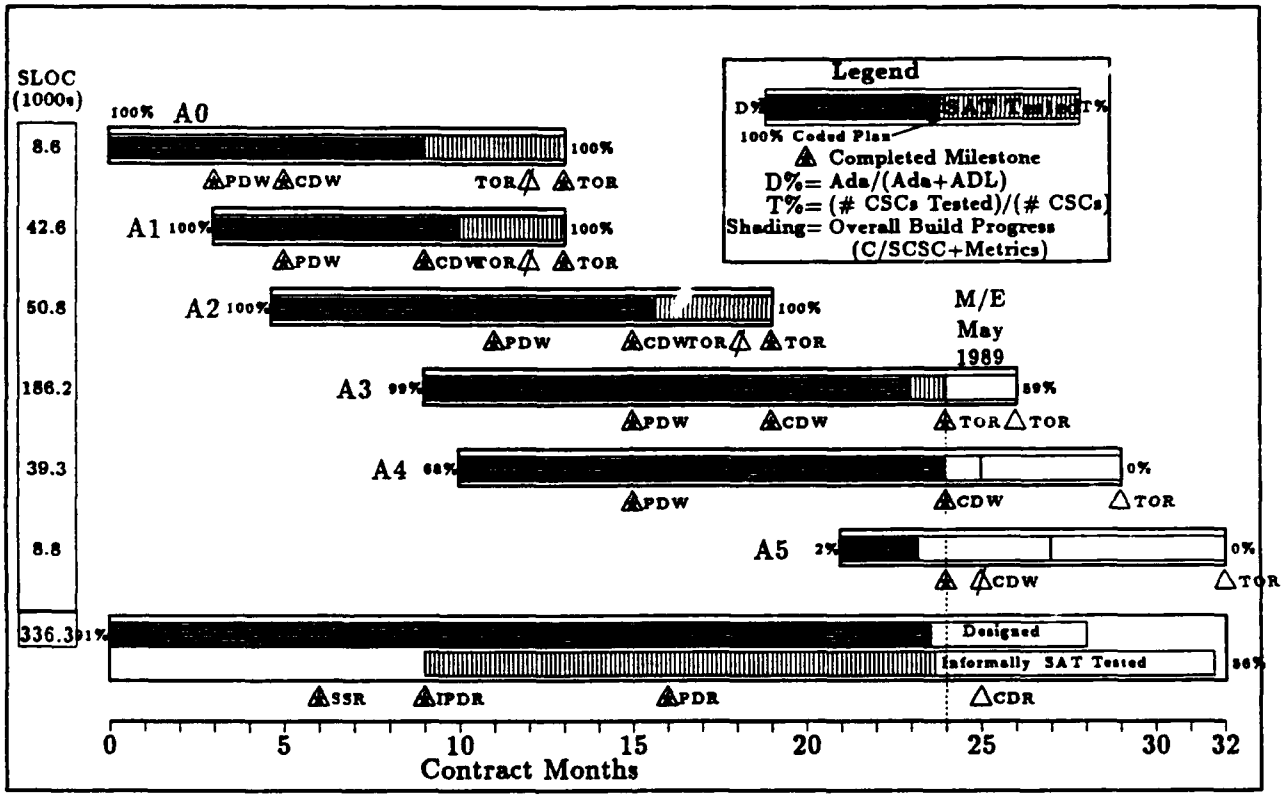


Figure 5: Development Progress-Common Subsystem

the status, by software CSCI for each build (Figure 6). The total number of software development files is tracked as well as the total estimated source lines of Ada code by CSCI. The designed/coded status is indicated by percent complete (Ada/(Ada+ADL)) of software source code. The current month (CM) column indicates the progress made during the past month by CSCI. This is helpful to determine which areas are falling behind or making little progress. The "tested" column shows informal stand alone testing complete as a percent of the total source lines of code. The "documented" status depicts the percent of software development files with completed documentation. The graphics depict the actual progress versus the plan for the total common subsystem software for designed, coded, standalone tested, and documented. This metric allows immediate insight into progress versus the plan. This same metric with build specific metrics (Figure 7) is used as the primary measurement by management to track the individual builds and is especially useful when many builds are in development simultaneously.

CSCI	Designed						Tested		Documented	
	Total SDFs	Total KSLOC	Complete (Ada)	TBD (ADL)	%	CM - ADL ⇒ Ada	Tested KSLOC	%	Complete SDFs	%
NAS	47	18.6	18.6	0	100%	-.2	18.6	100%	47	100%
SSV	46	182	182	0	100%	33.8	142.6	78%	22	47%
DCO	17	40.4	39.6	.8	98%	.7	25.8	63%	12	70%
TAS	16	9.6	8.3	1.3	86%	.7	9.6	100%	16	100%
CMP	21	10.2	10	.2	98%	-.3	3.8	37%	8	38%
CCO	9	75.5	57.8	17.7	76%	2.2	12.6	16%	2	22%
Total	156	336.3	316.3	20	94%	36.9	213	63%	107	68%

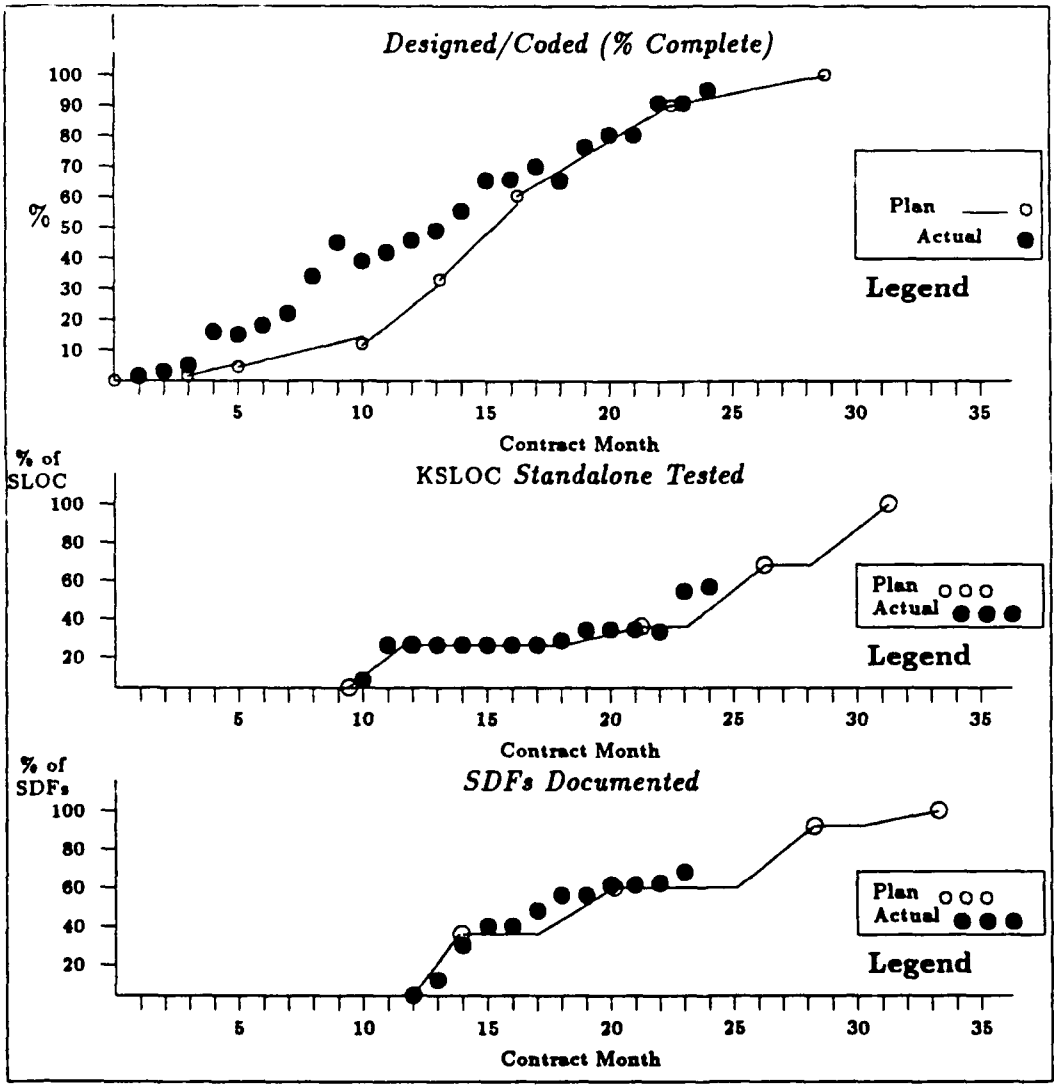


Figure 6: Development Progress-Common Subsystem

CSCI	Designed						Tested		Documented	
	Total SDFs	Total KSLOC	Complete (Ada)	TBD (ADL)	%	CM ADL ⇒ Ada	Tested KSLOC	%	Complete SDFs	%
NAS	6	1.5	1.5	0	100%	0	1.5	100%	6	100%
SSV	13	18	18	0	100%	.2	18	100%	13	100%
DCO	12	21.2	21.2	0	100%	-.1	21.2	100%	12	100%
TAS	6	2.8	2.8	0	100%	0	2.8	100%	6	100%
CMP	7	2.7	2.7	0	100%	0	2.7	100%	7	100%
CCO	2	4.6	4.6	0	100%	0	4.6	100%	2	100%
Total	46	50.8	50.8	0	100%	.1	50.8	100%	46	100%

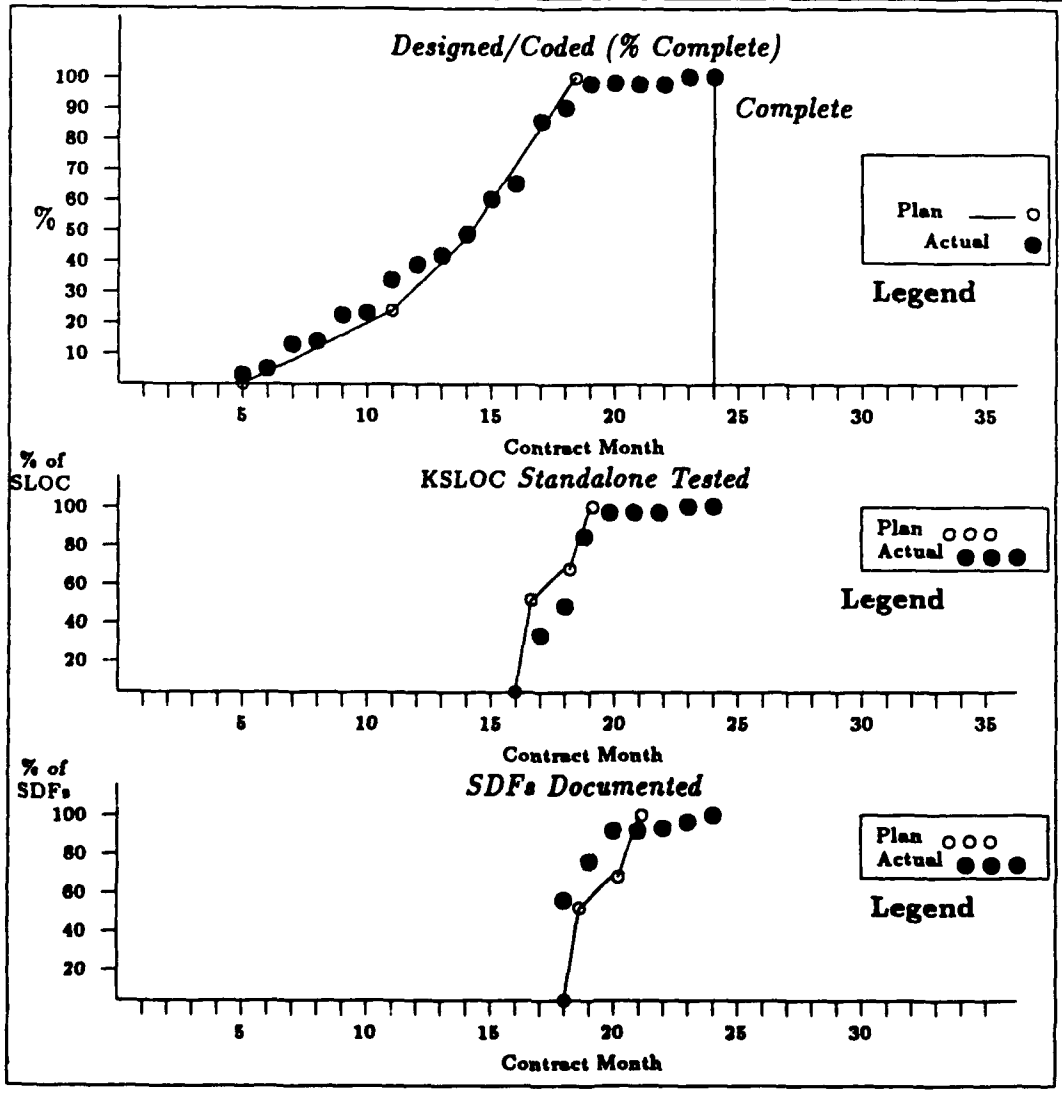


Figure 7: Development Progress-Build A2

Software staffing (Figure 8) is shown as personnel versus time and shows actual headcount as well as planned staffing needs. Additional metrics track attrition and additions on a monthly basis. A large turnover in personnel each month is a good indicator that downstream problems will occur due to the added training and "learning curve" required to assimilate new personnel. CCPDS-R's experience to date has been very positive with very low attrition other than planned departures. For example, the large change in Month 24 indicated the planned addition of test personnel and design engineers leaving the project.

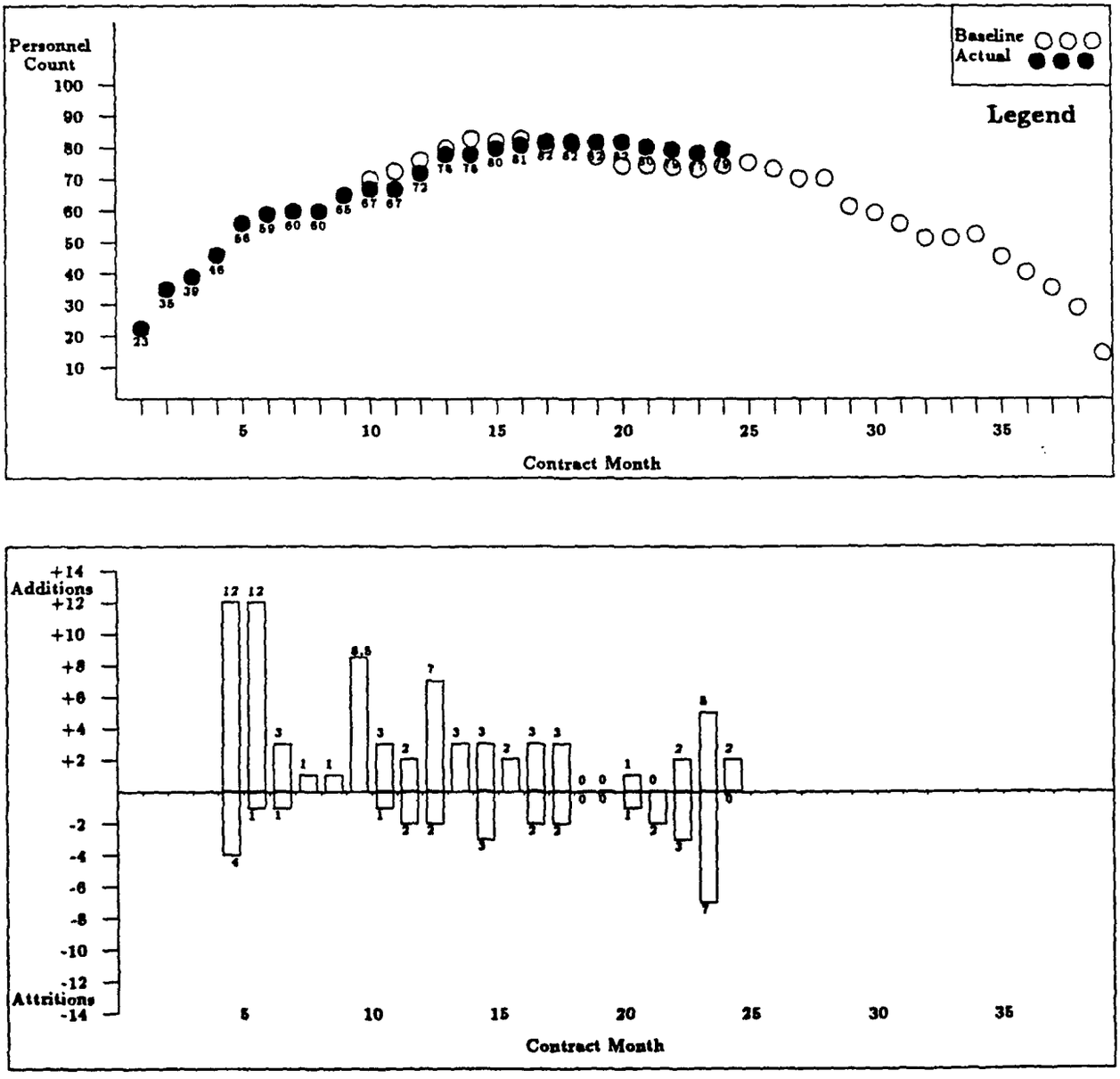


Figure 8: Staffing Profile History and Addition/Attrition History

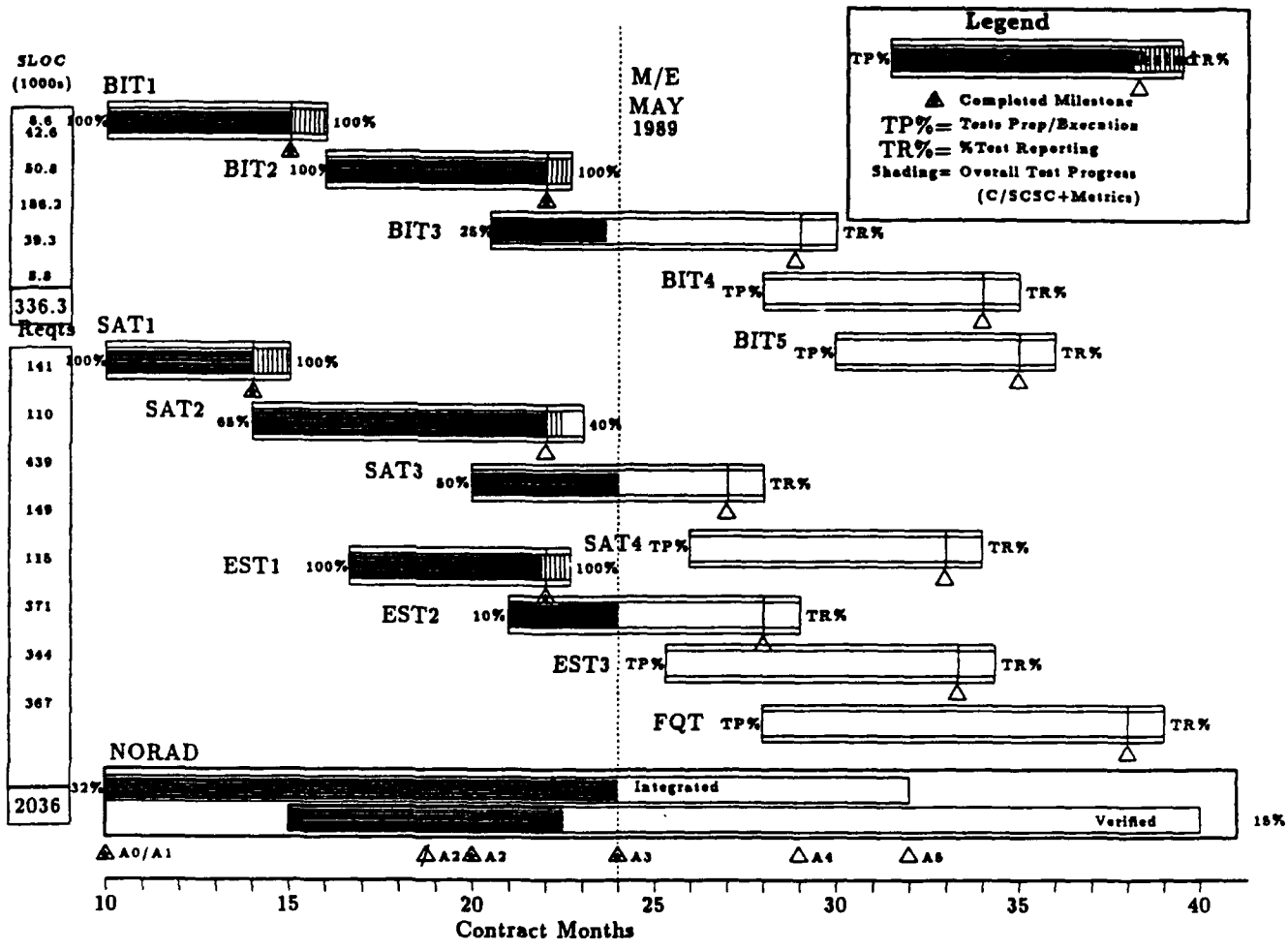


Figure 9: Testing Progress-Common Subsystem

The overall test progress for each subsystem is measured in a single graphic metric (Figure 9) similar to the development progress metric. This graphic is divided into two sections:

- Build Integration Testing (BIT)
- Formal Test including Stand Alone Testing (SAT), Engineering String Testing (EST) and Formal Qualification Testing (FQT)

The BIT progress is tracked for each software build by source lines of code and progress of test cases generated and executed. The measurement is percent complete; progress assessment is based on the monthly financial data. The Formal Test progress is tracked by requirements verified in each test period (SAT, EST, or FQT) and progress of test cases generated and executed. There is an overall status line to indicate the total subsystem progress as a percent integrated and as a percent verified.

The Software Requirement Verification metric (Figure 10) provides the detailed plan and status by test period and CSCI. This metric contains an indicator per CSCI per test period,

• Common Subsystem

pv = previously verified in SAT

Requirements Verified

TEST SOURCE	Completed/Total planned						
	NAS	SSV	DCO	TAS	CMP	CCO	TOTAL
Build A0/A1 SAT	48/49			5/5			53/54
Build A2 SAT		8/8	10/10	15/15	33/39	3/4	69/76
Build A3 SAT	14	126	58	14	56	4	0/272
Build A4 SAT			58		57	28	0/143
Build A5 SAT						5	0/5
EST 1	75/75		10/10	21/21			106/106
EST 2	59 (46 pv)	42	64	82 (1 pv)	6 (1 pv)	8	0/261 (48 pv)
EST 3	16	150	185	40	29	61	0/481
FQT	25	164	217	48	39	82	0/575
TOTAL	238	490	602	225	226	192	228/1973 (48 pv)

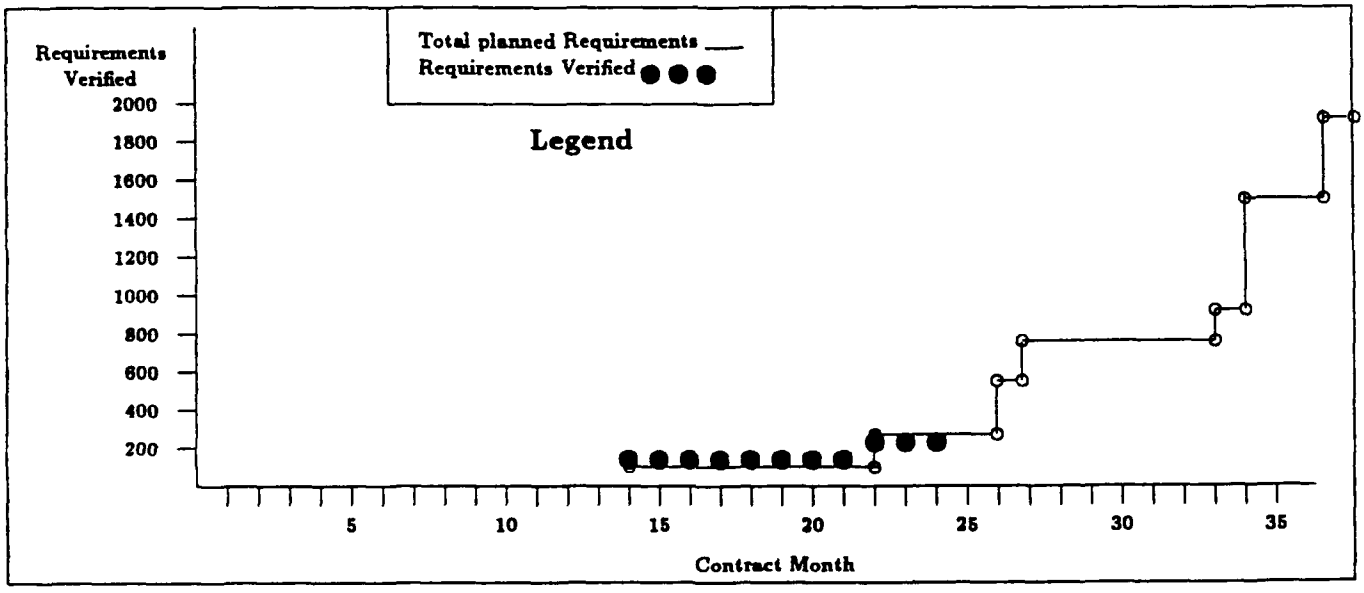


Figure 10: Verification Testing Progress

M/N where M is the number of requirements allocated to the test period and N is the number of requirements successfully verified and signed off by the government. The graphic depicts the progress versus the plan as a composite of the total requirements to be verified in the subsystem.

One of the benefits of the incremental development process is the ability to evaluate software quality and reliability early in the development cycle. A measure of this quality can be determined by the number of Software Problem Reports (SPRs) per CSCI (Figure 11). SPR statistics are gathered by CSCI, priority, and category. The open SPRs are addressed at the weekly Software Configuration Control Board (SCCB) to review status and approve appropriate action for resolution. The graphic metric tracks SPRs by age and is helpful to determine delinquent SPRs which may severely impact other areas of the program. We use a metric of SPRs/month scaled to 1000 Source lines of Ada code under test to illustrate software quality and reliability (Figure 12). The CCPDS-R average rate to date is .3 SPRs per 1000 SLOCs per month, which is well below TRW's experience on other major software projects at the equivalent stage of development.

CSCI	Priority					Category				Total	Open	Closed	Delinq
	1	2	3	4	5	S/W	Doc	Des	Other				
NAS	7	31	35	28	21	75	7	21	19	122	17	105	1
SSV	2	39	26	11	15	58	5	15	15	93	19	74	11
DCO	0	17	5	7	9	32	0	4	2	38	3	35	2
TAS	0	23	20	4	3	40	0	5	5	50	6	44	1
CMP	0	13	3	1	3	17	0	2	1	20	2	18	0
CCO	0	19	2	0	4	23	0	1	1	25	5	20	1
PSSV	0	0	0	0	0	0	0	0	0	0	0	0	0
PDCO	0	0	0	0	0	0	0	0	0	0	0	0	0
PTAS	0	0	0	0	0	0	0	0	0	0	0	0	0
PCO	0	0	0	0	0	0	0	0	0	0	0	0	0
Support	4	32	37	18	19	60	0	15	35	110	15	95	4
Test	0	23	6	2	0	12	2	0	17	31	1	30	1
OS/Vendor	1	3	27	14	13	24	0	0	34	58	12	46	3
Totals	14	200	161	85	87	341	14	63	129	547	80	467	24

Figure 11: Software Problem Reports (SPRs)

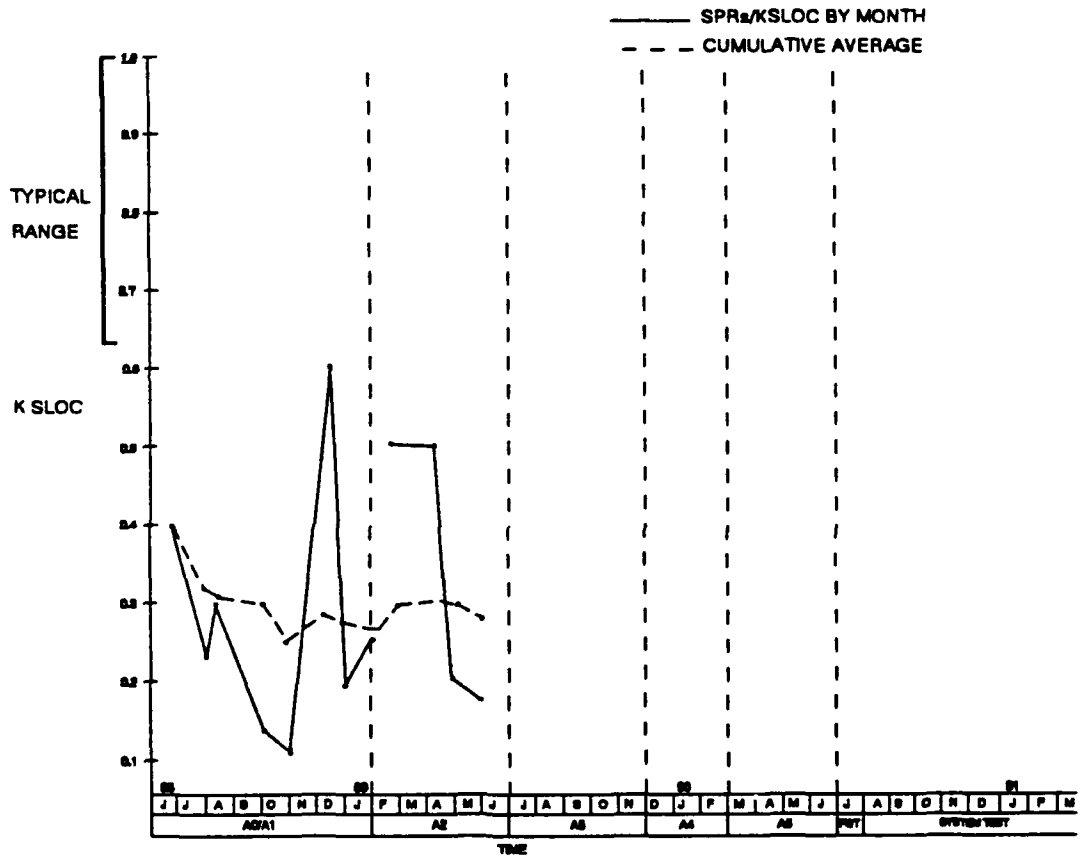


Figure 12: SPRs per KSLOC per Month

LESSONS LEARNED

The CCPDS-R program experience using the new Ada Process Model and software metrics has proven to be a successful. The project has reacted well to the new approach and adapted as necessary. Likewise, we have adapted the initial process model to provide greater efficiency and improved productivity. The following aspects of managing the software development process are working well:

- The early preparation and planning for the program. This was accomplished largely during the concept definition phase prior to the full-scale development and focused on standards, tools, schedules, staffing plans, test program plans and early development of the foundation components - NAS.
- The general approach based on the new process model. This includes the early software builds and design integration resulting in software being integrated largely prior to turnover to Integration and Test. The technical quality evaluation that is being performed by the software engineering and software test organizations augments the typical software quality assurance activities.
- The software metrics. The software metrics have proven to be effective management measurement tools for progress and quality assessment.
- The capability demonstrations. These demonstrations are timely, identify technical risk areas early and focus risk attention, force integration as part of the design process early, and make software visible to customers, users, and management.
- The early exercise of standards & tools. The development standards and tools were produced prior to the start of the full-scale development effort allowing the software team to hit the ground running at contract start. The first build pioneered the SSPM and tool set to incorporate lessons learned into the larger later builds.
- Adquate schedule for PDR and CDR. Scheduling adequate time for requirement definition and design prototying has helped to produce effective demonstrations at these major milestones thereby lowering the risk to the I&T schedule.
- Problem identification/resolution. The software team made numerous mistakes in the first 10 months of the project. We solved these problems early, flushed out the inefficiencies in the new process and substantially reduced rework time.

On the flip side, all is not perfect. There are some areas that can be improved as the project continues into the two new subsystems.

- Computer resources. Ada development projects require extensive computing resources (CPUs, memory, disks). This is especially true when the software is integrated into a large, testable entity. Making efficient use of these resources is a difficult task, requiring detailed planning for effective use of computer resources.
- Keep design out of the requirement specification. We were somewhat successful in this area; the project still has too much design detail in the software requirement specifications. [Grauling 1989]

- Everyone must follow Process Model: Designing and coding early requires total commitment. The government personnel and the project's system engineering organization must make timely decisions, especially as related to hardware and user interfaces. CCPDS-R uses government/contractor working groups to make these decisions, but timeliness could still improve.

- Automated Graphical representation of design. While this was not required by the software development team, it is a definite need by the reviewers of Ada design, such as, project management, system engineering, and government personnel. The development team has found that Ada itself is readable, unambiguous, and sufficient as a design/development methodology.

- Software Configuration Control. Our configuration control concept is based on a set of FORTRAN tools adapted to work in an Ada development environment. These are not well tuned to Ada and Ada compiler capabilities. We are presently investigating improvements in this area.

- Test program. The effectiveness of the test progress is yet to be determined. The approach to incremental testing seems to be working. The government imposed military standards and data item descriptions, which impose rigid constraints, can lead to a costly test program. [Springman 1989]

SUMMARY

The success of the CCPDS-R Program to date has been extraordinary. The software development team, all levels of management, and the customer/user community are very satisfied with our utilization of the new process model with progress metrics. Managers and customers appreciate the added insight into the software development process and the technical risk evaluation coupled with the approach. The software development and test team enjoy the opportunity to see their efforts demonstrated early in the process. This indirectly provides strong motivation to do well and has allowed them to have more time to focus on design and formal test, while minimizing the rudimentary software integration process. The user community is probably the most impressed since they have "seen" their ultimate product very early in the development cycle. This has permitted them to refine their "usability ideas" with actual hardware and software, as opposed to using paper concepts.

ACKNOWLEDGMENTS

The success of the CCPDS-R program to date is due to multiple contributions including TRW's System Engineering and Development Division's Management ability to try a new approach, ESD willingness to adapt to a new concept, and the entire CCPDS-R System and Software Engineering team to implement and produce within the given contractual constraints. Explicit acknowledgments are due to Walker Royce who was instrumental in developing the CCPDS-R process metrics and to Charlie Grauling, Tom Herman, and Mike Springman, whose day-to-day involvement, commitment, and management have guided the CCPDS-R program to achieve these successes.

BIOGRAPHY

Donald Andres is the Deputy Program Manager on the CCPDS-R Program. In this position he is responsible for all the software development activities from requirement definition through formal test. He received his B.S. in Mathematics from Syracuse University in 1968 and his M.S. in Computer Science from Purdue University in 1969. Mr. Andres has been at TRW for 17 years and has been in management positions for the past 14 years. His other assignments included Laboratory Manager for a group of 250 technical personnel performing software analysis and development activities on major Air Force Programs for avionics, command and control, and management information systems.

REFERENCES

[Grauling 1989] Grauling, C. G., "Requirements Analysis For Large Ada Programs: Lessons Learned on CCPDS-R", *TRI-Ada Proceedings*, Pittsburgh, October 1989.

[Springman 1989] Springman, M. C., "Incremental Software Test Methodology For A Major Government Ada Project ", *TRI-Ada Proceedings*, Pittsburgh, October 1989.

[Boehm 1985] Boehm, B. W., "The Spiral Model of Software Development and Enhancement", *Proceedings of the International Workshop on the software Process and Software Environments*, Coto de Caza, CA, March.

[Royce 1990] Royce, W.E., "TRW's Ada Process Model for Incremental Development of Large Software Systems", *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, March 26-30, 1990.

[Boehm/Royce 1988] Boehm, B. W., Royce W. E., "TRW IOC Ada COCOMO: Definition and Refinements", *Proceedings of the 4th COCOMO Users Group*, Pittsburgh, November 1988.

[Royce 1989] Royce, W. E., "Reliable, Reusable Ada Components for Constructing Large Distributed Multi-Task Networks: Network Architecture Services (NAS)", *TRI-Ada Proceeding*, Pittsburgh, October 1989.